Module 1

INTRODUCTION

- **Data** is a collection of facts and figures that can be processed to produce information.
- **Database** is a collection of related data, Database is one of the important components for many applications and is used for storing a series of data in a single set. In other words, it is a group / package of information that is put in order so that it can be easily access, manage and update. There are different types of database.
  They are:
  Bibliographic
  fulltext
  numeric
  images

In a database, even a smallest portion of information becomes the data. Example, Student is a data, roll number is a data, and address is a data, height, weight, marks everything is data. In brief, all the living and nonliving objects in this world is a data

- A **database management system** stores data in such a way that it becomes easier to retrieve, manipulate, and produce information. A **database-management system (DBMS)** is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the **database**, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*.

- **DBMS**
  A DBMS allows creation, definition and manipulation of database. DBMS is actually a tool used to perform any kind of operation on data in database. Dbms also provides protection and security to database. It maintains data consistency in case of multiple users. Here are some examples of popular dbms, MySql, Oracle, Sybase, Microsoft Access and IBM DB2 etc.

DATABASE-SYSTEM APPLICATIONS

- *Enterprise Information*

◦ ***Sales***: For customer, product, and purchase information.
◦ ***Accounting***: For payments, receipts, account balances, assets and other accounting information.
◦ ***Human resources***: For information about employees, salaries, payroll taxes, and benefits, and for generation of paychecks.
◦ ***Manufacturing***: For management of the supply chain and for tracking production of items in factories, inventories of items inwarehouses and stores, and orders for items.
◦ ***Online retailers***: For sales data noted above plus online order tracking, generation of recommendation lists, and maintenance of online product evaluations.

- *Banking and Finance*

◦ ***Banking***: For customer information, accounts, loans, and banking transactions.
◦ ***Credit card transactions***: For purchases on credit cards and generation of monthly statements.
◦ ***Finance***: For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm.

- *Universities*: For student information, course registrations, and grades (in addition to standard enterprise information such as human resources and accounting).

- *Airlines*: For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.

- *Telecommunication*: For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

STRUCTURED DATA	**SEMI STRUCTURED DATA**	**UNSTRUCTURED DATA**
• Strict format • All records have same format • Structured data is easily organized and generally stored in databases. • Structured data generally consists of numerical information and is objective • ATM, AIRLINE,SALES • Readily searchable, high degree of organization	• Data may have certain structure but not all information collected has same structure. • Semistructured data is data that is neither raw data, nor typed data in a conventional database system. It is structured data, but it is not organized in a rational model, like a table or an object based graph. • A lot of data found on the Web can be described as semistructured. • Data integration especially makes use of semistructured data.	• No predetermined form or structure • Very limited indication of data type • Compilation is a time and energy consuming task • Unstructured data can also come from social media sites such as Facebook, LinkedIn, Twitter, Tumblr, Flickr, Yelp, YouTube, and Pinterest. • Some examples of unstructured data include customer reviews that describe how they feel about an experience, identified triggers for readmission to hospital care, or call center conversations.

VIEW OF DATA

A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data. A major purpose of a database system is to provide users with an *abstract* view of the data. That is, the system hides certain details of how the data are stored and maintained.

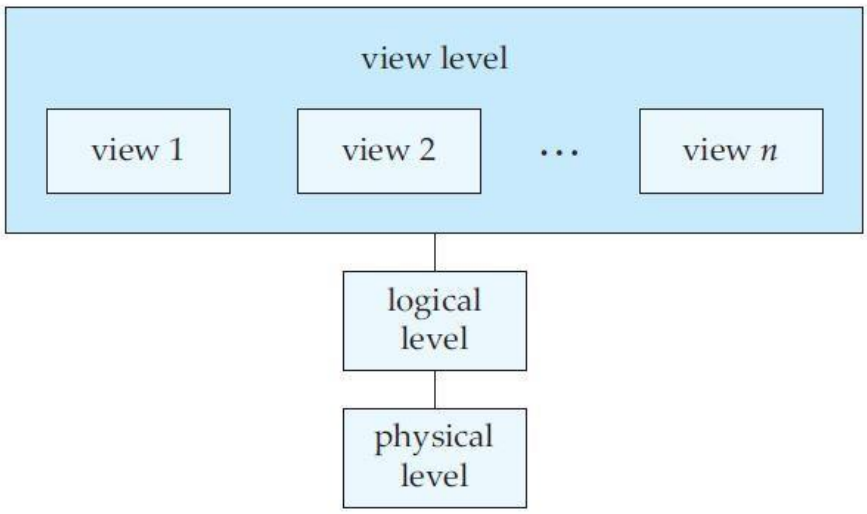
1. Data Abstraction

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system:

• **Physical level**. The lowest level of abstraction describes *how* the data are actually stored. The physical level describes complex low-level data structures in detail.

• **Logical level**. The next-higher level of abstraction describes *what* data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. This is referred to as **physical data independence**. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

• **View level**. The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.



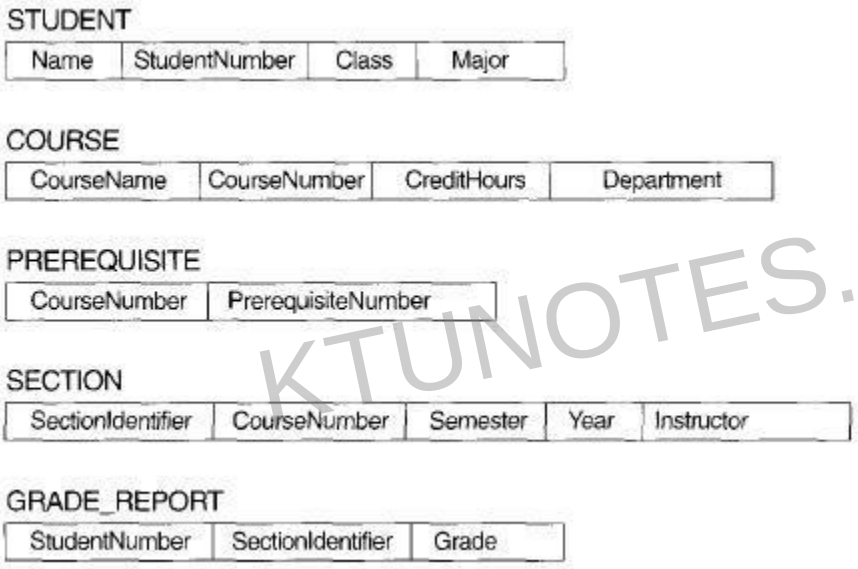
The three levels of data abstraction.

2. Instances and Schemas

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an **instance** of the database. Each variable has a particular value at a given instant. The values of the variables in a program at a point in time correspond to an *instance* of a database schema. Snapshot of the data in the database at a given instant time

The overall design of the database is called the database **schema**. A database schema corresponds to the variable declarations (along with associated type definitions) in a program. The description of a database is called the database schema, which is specified during database design and is not expected to change frequently. Blueprint of the Database how data is organized and relation among them are specified.

3. Schema Diagram



DATA MODELS

Underlying the structure of a database is the **data model**: a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. A data model provides a way to describe the design of a database at the physical, logical, and view levels.

There are a number of different data models. The data models can be classified into four different categories:

a. **Relational Model**. The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are also known as **relations**. The relational model is an example of a record-based model. Record-based models are so named because the

database is structured in fixed-format records of several types. Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes. The columns of the table correspond to the attributes of the record type. The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model.

b. **Entity-Relationship Model**. The entity-relationship (E-R) data model uses a collection of basic objects, called *entities*, and *relationships* among these objects. An entity is a "thing" or "object" in the real world that is distinguishable from other objects. The entity-relationship model is widely used in database design.

c. **Object-Based Data Model**.Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology. This led to the development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity. The object-relational data model combines features of the object-oriented data model and relational data model.

d. **Semistructured Data Model**. The semistructured data model permits the specification of data where individual data items of the same type may have different sets of attributes. This is in contrast to the data models mentioned earlier, where every data item of a particular type must have the same set of attributes. The **Extensible Markup Language (XML)** is widely used to represent semistructured data.

Historically, the **network data model** and the **hierarchical data model** preceded the relational data model. These modelswere tied closely to the underlying implementation, and complicated the task of modeling data. As a result they are used little now, except in old database code that is still in service in some places.

DATABASE LANGUAGES

A database system provides a **data-definition language** to specify the database schema and a **data-manipulation language** to express database queries and up dates. In practice, the data-definition and data-manipulation languages are not two separate languages; instead they simply form parts of a single database language, such as the widely used SQL language.

A. Data-Manipulation Language

A **data-manipulation language (DML)** is a language that enables users to access or manipulate data as organized by the appropriate data model. The types of access are:
• Retrieval of information stored in the database
• Insertion of new information into the database
• Deletion of information from the database
• Modification of information stored in the database

There are basically two types:
• **Procedural DMLs** require a user to specify *what* data are needed and *how* to get those data.
• **Declarative DMLs** (also referred to as **nonprocedural DMLs**) require a user to specify *what* data are needed *without* specifying how to get those data.

A **query** is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a **query language**. Although technically incorrect, it is common practice to use the terms *query language* and *data-manipulation language* synonymously.

B. Data-Definition Language

We specify a database schema by a set of definitions expressed by a special language called a **data-definition language** (**DDL**). The DDL is also used to specify additional properties of the data. DDL called a **data storage and definition** language. These statements define the implementation details of the database schemas, which are usually hidden from the users. The data values stored in the database must satisfy certain **consistency constraints**. For example, suppose the university requires that the account balance of a department must never be negative.
Thus, database systems implement integrity constraints that can be tested with minimal overhead:
• **Domain Constraints**. A domain of possible values must be associated with every attribute (for example, integer types, character types, date/time types). Declaring an attribute to be of a particular domain acts as a constraint on the values that it can take. Domain constraints are the most elementary form of integrity constraint. They are tested easily by the system whenever a new data item is entered into the database.

• **Referential Integrity**. There are cases where we wish to ensure that a value that appears in one relation for a given set of attributes also appears in a certain set of attributes in another relation (referential integrity). For example, the department listed for each course must be one that actually exists. More precisely, the *dept name* value in a *course* record must appear in the *dept name* attribute of some record of the *department* relation. Database modifications can cause violations of referential integrity.When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation.

• **Assertions**. An assertion is any condition that the database must always satisfy. Domain constraints and referential-integrity constraints are special forms of assertions. However, there are many constraints that we cannot express by using only these special forms. For example, "Every department must have at least five courses offered every semester" must be expressed as an assertion. When an assertion is created, the system tests it for validity. If the assertion is valid, then any future modification to the database is allowed only if it does not cause that assertion to be violated.

• **Authorization**. We may want to differentiate among the users as far as the type of access they are permitted on various data values in the database. These differentiations are expressed in terms of **authorization**, the most common being: **read authorization**, which allows reading, but

not modification, of data; **insert authorization**, which allows insertion of new data, but not modification of existing data; **update authorization**, which allows modification, but not deletion, of data; and **delete authorization**, which allows deletion of data. We may assign the user all, none, or a combination of these types of authorization.

DATABASE ADMINISTRATORS

In any organization where many persons use the same resources, there is a need for a chief administrator to oversee and manage these resources. In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software. Administering these resources is the responsibility of the database administrator (DBA). The DBA is responsible for **authorizing access to the database, for coordinating and monitoring its use, and for acquiring software and hardware resources as needed. The DBA is accountable for problems such as breach of security or poor system response time.** In large organizations, the DBA is assisted by a staff that helps carry out these functions.

DATABASE DESIGNERS

Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data. These tasks are mostly undertaken before the database is actually implemented and populated with data. It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements, and to come up with a design that meets these requirements.

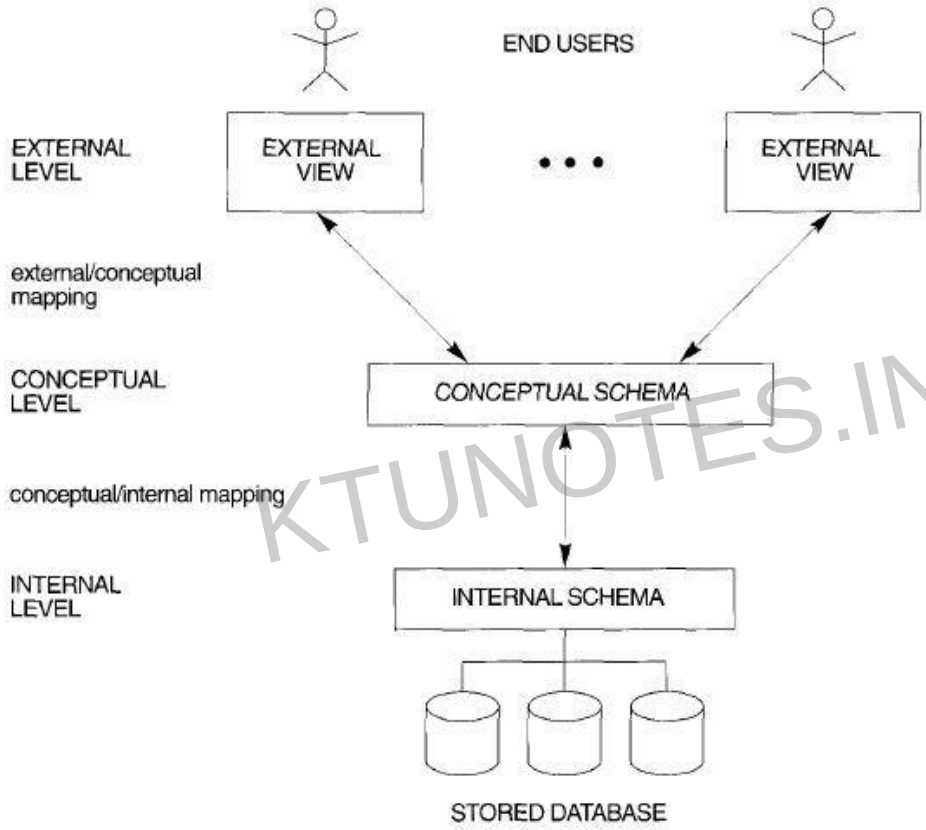
END USERS

End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use. There are several categories of end users:

- **Casual end users** occasionally access the database, but they may need different information each time. They use a sophisticated database query language to specify their requests and are typically middle- or high-level managers or other occasional browsers.
- **Naive or parametric end users** make up a sizable portion of database end users. Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates-called canned transactions-that have been carefully programmed and tested. The tasks that such users perform are varied: Bank tellers check account balances and post withdrawals and deposits. Reservation clerks fur airlines, hotels, and car rental companies check availability for a given request and make reservations. Clerks at receiving stations for courier mail enter package identifications via bar codes and descriptive information through buttons to update a central database of received and in-transit packages.
- **Sophisticated end users** include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS so as to implement their applications to meet their complex requirements.
- **Stand-alone users** maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces. An example is the user of a tax package that stores a variety of personal financial data for tax purposes

THREE-SCHEMA ARCHITECTURE

This explains the database system organization, A three-schema architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS. Three of the four important characteristics of the database approach,
(1) insulation of program:; and data (program-data and program-operation independence),
(2) support of multiple user views, and
(3) use of a catalog to store the database description (schema).
Three-schema architecture was proposed to help achieve and visualize these characteristics.



The three-schema architecture.

1. **The internal level** has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database. How data is stored, space allocation, encryption, record placement etc..

2. **The conceptual level** has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. Usually, a representational data model is used to describe the conceptual schema when a database system is implemented. This *implementation conceptual schema* is often based

on a *conceptual schema design* in a high-level data model. Relationship between data, Constraints etc..

3. **The external or view level** includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. As in the previous case, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high level data model.

DATA INDEPENDENCE

Upper levels are not affected by change in lower level. The three-schema architecture can be used to further explain the concept of data independence, which can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

PHYSICAL DATA INDEPENDENCE	LOGICAL DATA INDEPENDENCE
• Physical storage or devices can be changed without affecting conceptual schema • Modification at this level is to improve performance • Not difficult (just changing the location of data)	• Conceptual schema can be changed without affecting the external schema • Structure of database is altered in this level • Difficult (depends on the affecters)

1. **Logical data independence** is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item). In the last case, external schemas that refer only to the remaining data should not be affected.

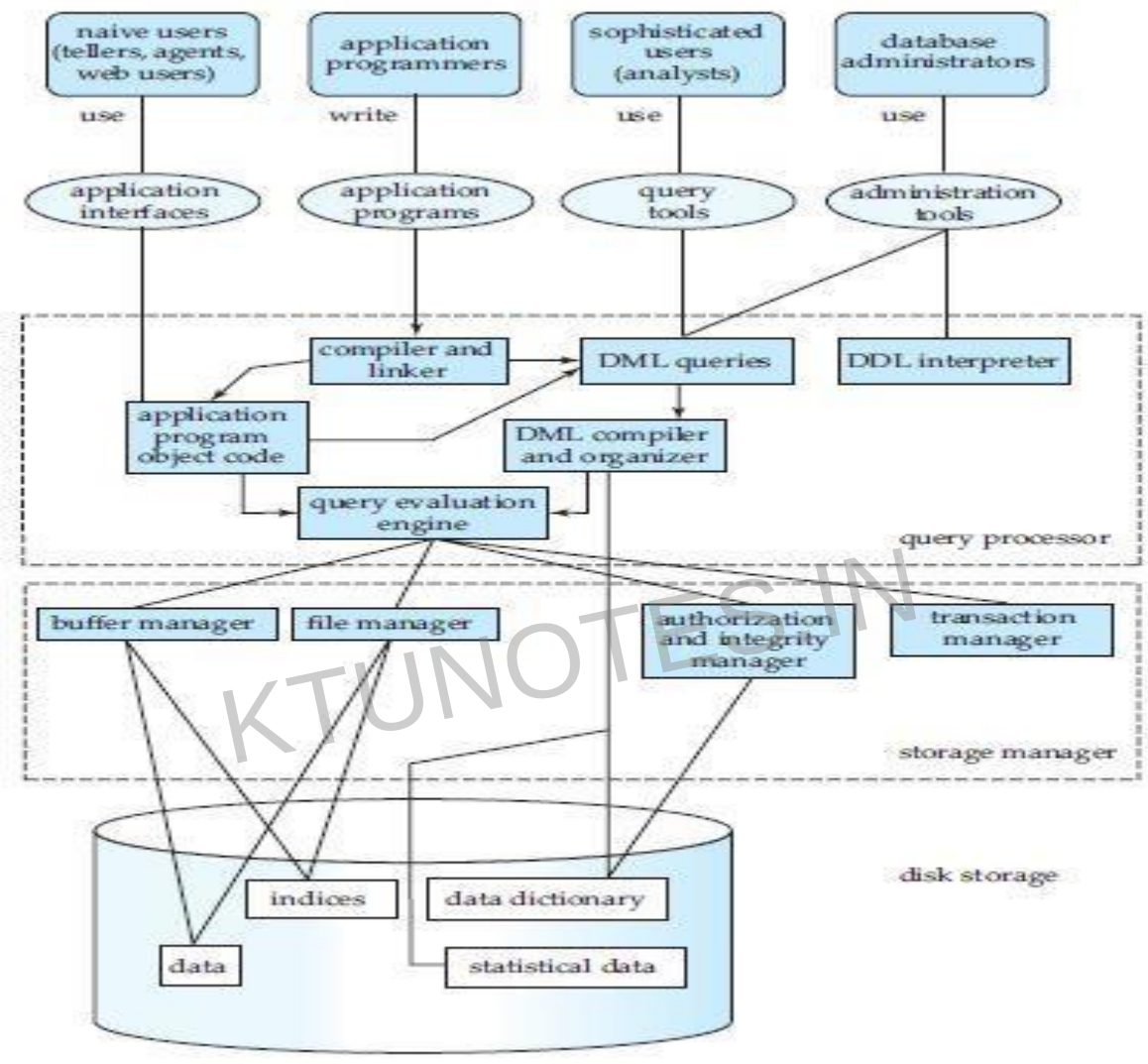
2. **Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well. Changes to the internal schema may be needed because some physical files had to be reorganized-for example, by creating additional access structures-to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema.

ARCHITECTURES FOR DBMS

The architecture of a database system is greatly influenced by the underlying computer system on which the database system runs. Database systems can be centralized, or client-server, where one server machine executes work on behalf of multiple client machines. Database systems can also be designed to exploit parallel computer architectures. Distributed databases span multiple geographically separated machines.

Most users of a database system today are not present at the site of the database system, but connect to it through a network. We can therefore differentiate between **client** machines, on which remote database users work, and **server** machines, on which the database system runs.

Database applications are usually partitioned into two or three parts, as in Figure below

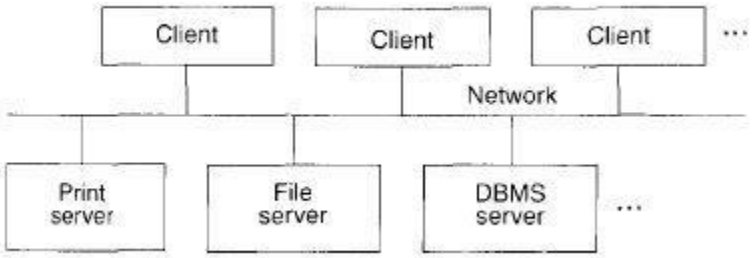


System structure.

The client/server architecture was developed to deal with computing environments in which a large number of rcs, workstations, file servers, printers, database servers, Web servers, and other equipment are connected via a network. The idea is to define specialized servers with specific functionalities. A client in this framework is typically a user machine that provides user interface capabilities and local processing. When a client requires access to additional functionality-such as database access-that does not exist at that machine, it connects to a server that provides the needed functionality. A server is a machine that can provide services to the client machines, such as file access, printing, archiving, or database access.

1. TWO-TIER CLIENT/SERVER ARCHITECTURES FOR DBMS

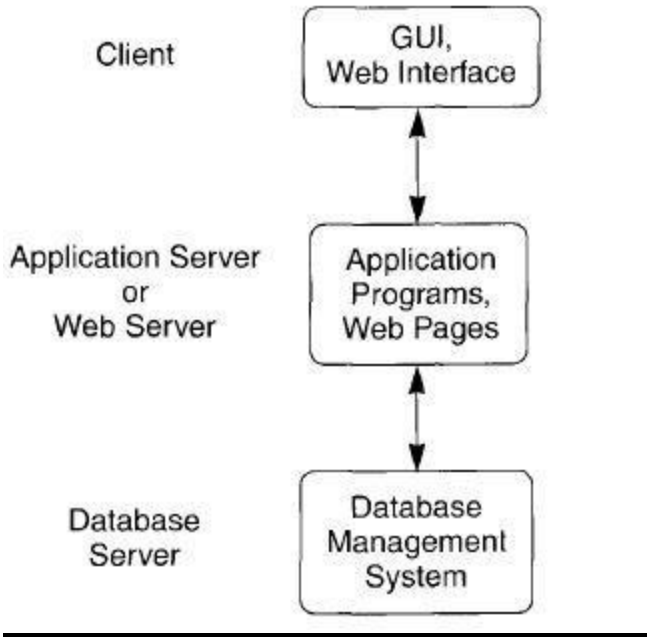
In relational DBMSs (RDBMSs), many of which started as centralized systems, the system components that were first moved to the client side were the user interface and application programs. In such an architecture, the server is often called a query server or transaction server, because it provides these two functionalities. In RDBMSs, the server is also often called an SQL server, since most RDBMS servers are based on the SQL language and standard.



When DBMS access is required, the program establishes a connection to the DBMS (which is on the server side); once the connection is created, the client program can communicate with the DBMS. A standard called Open Database Connectivity (ODBC) provides an application programming interface (API), which allows client-side programs to call the DBMS, as long as both client and server machines have the necessary software installed. A related standard for the Java programming language, called JDBC, has also been defined. This allows Java client programs to access the DBMS through a standard interface.

2. THREE-TIER CLIENT/SERVER ARCHITECTURES FOR WEB APPLICATIONS

Many Web applications use an architecture called the three-tier architecture, which adds an intermediate layer between the client and the database server. This intermediate layer or middle tier is sometimes called the application server and sometimes the Web server, depending on the application. This server plays an intermediary role by storing business rules (procedures or constraints) that are used to access data from the database server. It can also improve database security by checking a client's credentials before forwarding a request to the database server. Clients contain GUI interfaces and some additional application-specific business rules. The intermediate server accepts requests from the client, processes the request and sends database commands to the database server, and then acts as a conduit for passing (partially) processed data from the database server to the clients, where it may be processed further and filtered to be presented to users in GUI format. Thus, the *user interface*, *application rules*, and *data access* act as the three tiers. Advances in encryption and decryption technology make it safer to transfer sensitive data from server to client in encrypted form, where it will be decrypted. The latter can be done by the hardware or by advanced software. This technology gives higher levels of data security.



CLASSIFICATION OF DATABASE MANAGEMENT SYSTEMS

DBMS is classified according to several criteria: data model, number of users, number of sites, cost, types of access paths, and generality. The main classification of DBMSs is based on the data model.

1. Data model

The main data model used in many current commercial DBMSs is the relational data model. The object data model was implemented in some commercial systems but has not had widespread use. Many legacy (older) applications still run on database systems based on the hierarchical and network data models. The relational DBMSs are evolving continuously, and, in particular, have been incorporating many of the concepts that were developed in object databases. This has led to a new class of DBMSs called object-relational DBMSs. We can hence categorize DBMSs based on the data model: relational, object, object-relational, hierarchical, network, and other.

2. Number of users

Single-user systems support only one user at a time and are mostly used with personal computers. Multiuser systems, which include the majority of DBMSs, support multiple users concurrently.

3. Number of sites

DBMS is centralized if the data is stored at a single computer site. A centralized DBMS can support multiple users, but the DBMS and the database themselves reside totally at a single computer site. A distributed DBMS (DDBMS) can have the actual database and DBMS software distributed over many sites, connected by a computer network. Homogeneous DDBMSs use the same DBMS software at multiple sites. A recent trend is to develop software to access several

autonomous preexisting databases stored under heterogeneous llBMSs. This leads to a federated DBMS (or multidatabase system), in which the participating DBMSs are loosely coupled and have a degree of local autonomy. Many llDBMSs use a client-server architecture.

4. **Types of access path**

DBMS can be general purpose or special purpose. When performance is a primary consideration, a special-purpose DBMS can be designed and built for a specific application; such a system cannot be used for other applications without major changes. Many airline reservations and telephone directory systems developed in the past are special purpose DBMSs. These fall into the category of online transaction processing (OLTP) systems, which must support a large number of concurrent transactions without imposing excessive delays.

5. **Cost**

****